D. Etiemble

ARCHITECTURE DES ORDINATEURS

Corrigé Examen Décembre 2012
2 H – Tous documents autorisés
Les questions sont indépendantes

1. Nombres flottants

Soit le programme C qui calcule de trois manières différentes la somme des N premiers entiers (partie gauche de la Figure 1)

```
main () {
int i; double x, x1, x2,x3;
x= N * (N-1.0)/2.0;
printf ("sum0= %f \n",x);

x=0.0;
for (i=0; i<N; i++)
   x+= i;
printf ("sum1= %f \n",x);

x=0.0 ;x1=0.0 ; x2=0.0 ; x3=0.0;
for (i=0; i<N; i+=4)
{
   x+=i;
   x1+=i+1;
   x2+=i+2;
   x3+=i+3; }
x+=x1+x2+x3;
printf ("sum4= %f \n",x);
}
```

```
sum0= 124999999750000000.000000
sum1= 124999999567108900.000000
sum4= 124999999750000000.000000
Appuyez sur une touche pour continuer...
```

sum4= 124999999750000000.000000 Appuyez sur une touche pour continuer...

Figure 1 : Programme C (gauche) et résultats (droite).

Q 1) Expliquer les résultats d'exécution obtenus (partie droite de la figure 1).

Les versions sum0 et sum4 donnent un résultat exact (calcul de la somme des N premiers entiers de 0 à N-1). La version sum1 donne un résultat faux.

La raison est que la version sum4 découpe les sommations en 4 parties qui permettent des additions exactes avec des « doubles » alors que la somme obtenue avec la version sum1, approximativement 4 fois plus grande que les sommes partielles de Sum4, introduit des erreurs d'arrondi lors des accumulations pour les grandes valeurs de i.

(NB : ce n'est pas un problème de « cast » de 1 vers 1.0, mais des problèmes d'arrondi).

2. Pipeline

On suppose qu'un processeur a les pipelines suivants :

Instructions entières UAL : 5 étages
LI DI/LR EX MEM ER
Instructions de chargement flottant (LF) : 5 étages
LI DI/LR EX MEM EF
Instructions flottantes (addition ou multiplication) : 7 étages
LI DI LF EX1 EX2 EX3 EF

avec la signification suivante :
LI : lecture des instructions dans le cache instructions
DI : décodage des instructions
LR : lecture registres entiers
LF : lecture des registres flottants
EX : exécution UAL pour les entiers, et calcul des adresses (mémoire et branchements)
EXi : phase d'une exécution flottante
MEM : accès au cache des données
ER : Écriture dans les registres entiers.
EF : Ecriture dans les registres flottants
Tous les "bypass" nécessaires existent.

On rappelle que la latence est définie par le nombre de cycles entre une instruction qui produit un résultat et l'instruction qui utilise ce résultat.
On rappelle que le délai de branchement est le nombre de cycles entre l'instruction de branchement et l'instruction cible du branchement. Si le branchement est au cycle i, et l'instruction cible est au cycle i+n, alors le délai de branchement est n-1.

Q 2) Donner les latences pour les cas suivants :

a) latence d'une instruction UAL entière lorsqu'elle est suivie d'une autre instruction UAL ?

LI LR **EX** MEM ER
LI LR **EX** MEM ER
Latence = 1

b) latence d'une instruction de chargement de nombre flottant (LF) lorsqu'elle est suivie par une opération flottante (addition ou multiplication) ?

LI LR EX **MEM** EF
LI DI LF **EX1** EX2 EX3 EF
Latence = 1

c) latence d'une instruction flottante (addition ou multiplication) lorsqu'elle est suivie par une autre instruction flottante ?

LI DI LF EX1 EX2 **EX3** EF
LI DI LF **EX1** EX2 EX3 EF
Latence = 3

d) Latence et délai de branchement pour les instructions du type BNE ?

LI LR **EX** MEM ER
LI LR EX MEM ER
Latence = 3
Délai de branchement = 2

e) Latence et délai de branchement des instructions de type JR ?
LI **LR** EX MEM ER
LI LR EX MEM ER
Latence = 2
Délai de branchement = 1

3. Optimisation de programmes

On utilise un processeur scalaire dont les instructions sont définies dans l'annexe 1 (Table 2 et Table 3). Les latences des instructions sont données dans la troisième colonne des tables. On rappelle qu'une latence de n signifie que si une instruction I démarre au cycle n, une instruction qui utilise son résultat ne peut démarrer qu'au cycle c+n (une latence de 1 signifie qu'elle peut démarrer au cycle suivant). Les sauts et branchements ne sont pas retardés et l'on suppose une prédiction de branchement parfaite (l'instruction BNE a une latence de 1 cycle).

La table 1 présente un programme C et le programme assembleur correspondant. Les tableaux X, Y et Z sont rangés successivement à partir de l'adresse 0x1000 0000, qui est contenue dans le registre R3 au démarrage du programme.

float X[100], Y[100], Z[100], A , B;
int i;
 	
for (i=0; i<100; i++)
Z[i]= A*X[i] + B*Y[i];
	ADDI R5,R3,400
	LF F1, A //A est chargé dans F1
	LF F2,B //B est chargé dans F2
	Loop : LF F3,0(R3)
	LF F4, 400(R3)
	FMUL F3,F3,F1
	FMUL F4,F4,F2
	FADD F3,F3,F4
	SF F3,800(R3)
	ADDI R3,R3,4
	BNE R3,R5,Loop

Table 1 : Programme C et programme assembleur.

Q 3) Donner l'exécution cycle par cycle de la boucle optimisée (mais sans déroulage de boucle). Quel est le nombre de cycles par itération de la boucle ? Quel est le temps total d'exécution du programme ?

	LF F1,A
	LF F2,B
	ADDI R5,R3,400
1 - Loop	LF F3,0(R3)
2	LF F4, 400(R3)

3	FMUL F3,F3,F1
4	FMUL F4,F4,F2
5	ADDI R3,R3,4
6	
7	FADD F3,F3,F4
8	
9	
10	SF F3,796(R3)
11	BNE R3,R5,Loop

11 cycles par itération.
Temps d'exécution total : 3+11*100=1103

Q 4) Reprendre la question Q3) avec un déroulage de boucle d'ordre 4.

	LF F1,A
	LF F2,B
	ADDI R5,R3,400
1 - Loop	LF F3,0(R3)
2	LF F5, 4(R3)
3	LF F7, 8(R3)
4	LF F9,12(R3)
5	LF F4, 400(R3)
6	LF F6,404(R3)
7	LF F8, 408(R3)
8	LF F10,412(R3)
9	FMUL F3,F3,F1
10	FMUL F4,F4,F2
11	FMUL F5,F5,F1
12	FMUL F6,F6,F2
13	FMUL F7,F7,F1
14	FMUL F8,F8,F2
15	FMUL F9,F9,F1
16	FMUL F10,F10,F2
17	FADD F3,F3,F4
18	FADD F5,F5,F6
19	FADD F7,F7,F8
20	FADD F9,F9,F10
21	SF F3,800(R3)
22	SF F5,804(R3)
23	SF F7,808(R3)
24	SF F9,812(R3)
25	ADDI R3,R3,16
26	BNE R3,R5,Loop

26/4 = 6,5 cycles/itération.
Temps d'exécution totale : 3 + 200 * 26 = 5203 cycles

4. Caches

Un processeur utilise un cache de données de 16 Ko, avec des lignes de 16 octets, à correspondance directe. Le cache utilise la réécriture avec écriture allouée (il y a des défauts de cache en écriture). Le processeur a des adresses sur 32 bits.

On considère l'extrait de programme C suivant, pour lequel les tableaux X et Y sont rangés successivement en mémoire à partir de l'adresse 1000 0000_H (adresse de X[0].)

```
float X[4096], Y[2048];
for (i=0 ; i<2048 ; i++)
      Y[i] = X[i+2048] – X[i] ;
```

Q 5) Quel est le nombre de bits pour l'adresse dans la ligne, l'index et l'étiquette ?

Adresse dans la ligne : 4 bits
Index : 10 bits (1K lignes)
Etiquette : 32-14 = 18 bits

Q 6) Quelles sont les adresses hexadécimales de Y[0] et X[2048] ? Dans quelles lignes du cache vont les données correspondant à X[0], X[2048] et Y[0] ?

Déplacement X[2048] = 2048 * 4 = 8192 = 2000H
Adresse X[2048] = 1000 2000H = 0001 0000 0000 0000 00|10 0000 0000 |0000
Adresse Y[0] = 1000 4000H = 0001 0000 0000 0000 01|00 0000 0000| 0000
X[0] et Y[0] vont dans la ligne 0 du cache
X[2014] va dans la ligne 512

Q 7) Quel est le nombre de défauts de cache par itération de la boucle et le nombre total de défauts de cache

Avec la correspondance directe, ce programme a

- 2 défauts de cache par itération pour X[0] et Y[0]
- 1 défaut toutes les 4 itérations pour X[2048]
- Soit 2,25 défauts par itération et un total de 4608 défauts.
-

Soit un déroulage de 4 de la boucle initiale.

```
float X[4096], Y[2048];
for (i=0 ; i<2048 ; i+=4){
      Y[i] = X[i+2048] – X[i] ;
      Y[i+1] = X[i+2049] – X[i+1] ;
      Y[i+2] = X[i+2050] – X[i+2] ;
      Y[i+3] = X[i+2051] – X[i+3] ;}
```

Q 8) Proposer un ordre d'accès aux tableaux qui minimise le nombre de défauts de cache. Quel est alors le nombre total de défauts de cache ?

Données de type float

Il faut utiliser une ligne de cache complètement avant qu'elle soit éjectée

Une ligne de cache contient 4 floats. Avec un déroulage de boucle d'ordre 4 et en accédant d'abord à X[i], X[i+1], X[i+2], X[i+3] avant d'accéder à X[i+2048], X[i+2049], X[i+2050], X[i+2051], puis Y[0], Y[1],

Y[2], Y[3], alors il y a 3 défauts de cache pour 4 itérations (0.75 défaut par itération) au lieu de 2,25 défauts par itération, soit un total de 0,75 *2048 = 1536 défauts.

Données de type double.

Il y a deux doubles par ligne de cache.

Déplacement X[2048] = 2048 * 8 = 16384 = 4000H
Adresse X[2048] = 1000 4000H = 0001 0000 0000 0000 01|00 0000 0000 |0000
Adresse Y[0] = 1000 8000H = 0001 0000 0000 0000 10|00 0000 0000| 0000
X[0], Y[0] et X[2048] vont dans la ligne 0 du cache.
Il y a un défaut de cache à chaque accès soit 2048 * 3 = 6144 défauts.
Dans ce cas, un déroulage de boucle d'ordre 2, avec accès en accédant d'abord à X[i], X[i+1], avant d'accéder à X[i+2048], X[i+2049 puis Y[0] Y[1] permet d'avoir 3 défauts de cache toutes les deux itérations soit 1,5 défauts par itération.

5. LOI D'AMDAHL

Un programme séquentiel a 25% de son temps d'exécution qui ne peut être parallélisé. On veut l'accélérer à taille constante.

Q 9) Quel est le nombre de processeurs nécessaire pour obtenir une efficacité parallèle de 50% ? (Efficacité parallèle = Accélération / Nombre de processeur).

EP = Acc/n = $\dfrac{1}{n*(0{,}25+\dfrac{0{,}75}{n})} = 0{,}5$

0,25 n + 0,75 = 2
0,25 n= 1,25
n= 1,25 *4 = 5

6. Annexe 1

JEU D'INSTRUCTIONS (extrait)

Mnémo	Syntaxe	Latence	Effet
ADDI	ADDI Rd, Ra, IMM	1	Rd ←Ra + IMM-16 bits avec ES
BEQ	BEQ Ri, Rj, dépl	1	si Ri=Rj alors CP ← NCP + depl
BGE	BNE Ri, Rj, dépl	1	si Ri≠Rj alors CP ← NCP + depl

Table 2 : instructions entières disponibles

Mnémo	Syntaxe	Latence	Effet
LF	LF Fi, dép.(Ra)	2	Fi ← M (Ra + dépl.16 bits avec ES)
SF	SF Fi, dép.(Ra)	1	Fi -> M (Ra + dépl.16 bits avec ES)
FADD	FADD Fd, Fa, Fb	3	Fd ←Fa + Fb
FMUL	FMUL Fd, Fa, Fb	3	Fd ←Fa x Fb

Table 3 : Instructions flottantes